`

LAB 03

DATA TYPES & ASSEMBLY INSTRUCTIONS

EL 2003 COMP ORG & ASSEMBLY LANGUAGE LAB

LABORATORY MANUAL Fall 2021Instructor: Aamir Ali, Aashir Mehboob, Amin Sadiq, Qurat-ul-Ain, M Kariz, Rabia

Syed Muhammad Faheem	20K-1054	3E
STUDENT NAME	ROLL NO	SEC

____
SIGNATURE & DATE

MARKS AWARDED: ______________

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES (NUCES), KARACHI

Prepared by: Amin Sadiq

Version: 1.0

Date: 17th Sep 2021

Lab Session 03 DATA TYPE & ASSEMBLY INSTRUCTIONS

~~Lab Session 03: DATA TYPE & ASSEMBLY INSTRUCTIONS~~

Objectives:

- Defining Data
- Data Definition Statement
- Data Initializations
- Multiple Initializations
- String Initialization
- Assembly language Instructions: MOV , ADD , SUB
- Sample Program
- Exercise

Data Types:

MASM defines **intrinsic data types**, each of which describes a set of values that can be assigned to variables and expressions of the given type.

BYTE	8-bit unsigned integer
SBYTE	8-bit signed integer. S stands for signed
WORD	16-bit unsigned integer
SWORD	16-bit signed integer
DWORD	32-bit unsigned. D stands for double
SDWORD	32-bit signed integer
QWORD	64-bit integer. Q stands for quad
TBYTE	80-bit integer. T stands for ten

Data definition statement:

A data definition statement sets aside storage in memory for a variable, with an optional name.

Data definition statements create variables based on intrinsic data types.

A data definition has the following syntax:

[name] directive initializer [,initializer]...

Initializer: At least one initializer is required in a data definition, even if it is zero. Additional initializers, if any, are separated by commas. For integer data types, initializer is an integer constant or expression matching the size of the variable's type, such as BYTE or WORD. If you prefer to leave the variable uninitialized (assigned a random value), the ? symbol can be used as the initializer.

Examples:

value1 **BYTE** 'A' ; character constant value2 **BYTE** 0 ; smallest
unsigned byte value3 **BYTE** 255 ; largest unsigned byte value4
SBYTE −128 ; smallest signed byte value5 **SBYTE** +127 ; largest
signed byte

greeting1 **BYTE** "Good afternoon", 0 ; String constant with null terminated string
greeting2 **BYTE** 'Good night'; String constant greeting1 **BYTE** 'G','o','o','d' ;
String constant

The hexadecimal codes 0Dh and 0Ah are alternately called CR/LF (carriage-return line-feed) or end-of-line characters.

list BYTE 10,20,30,40 ; Multiple initializers

Note: A question mark (?) initializer leaves the variable uninitialized, implying it will be assigned

a value at runtime: value6 BYTE ?

DUP Operator

The DUP operator allocates storage for multiple data items, using a constant expression as a counter. It is particularly useful when allocating space for a string or array, and can be used with initialized or uninitialized data.

Examples:

v1 BYTE 20 DUP(0) ; 20 bytes, all equal to zero v2 BYTE 20
DUP(?) ; 20 bytes, uninitialized

v3 BYTE 4 DUP("STACK") ;20 bytes, "STACKSTACKSTACKSTACK"

Operand Types:

As x86 instruction formats:

[label:] mnemonic [operands][; comment]

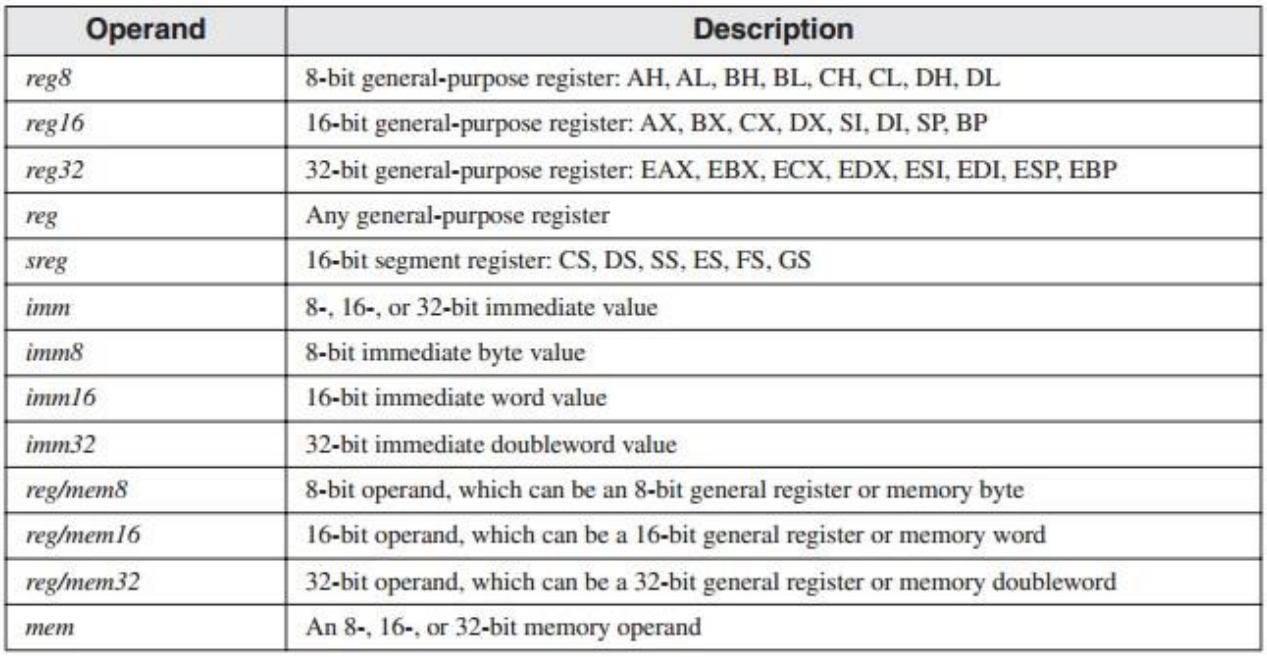
Because the number of operands may vary, we can further subdivide the formats to have zero, one, two, or three operands.
Here, we omit the label and comment fields for clarity:

mnemonic
mnemonic [destination] mnemonic
[destination],[source] mnemonic
[destination],[source-1],[source-2]

x86 assembly language uses different types of instruction operands. The following are the easiest to use:

- Immediate—uses a numeric literal expression
- Register—uses a named register in the CPU
- Memory—references a memory location

Following table lists a simple notation for operands. We will use it from this point on to describe the syntax of individual instructions.



Operand	Description
reg8	8-bit general-purpose register: AH, AL, BH, BL, CH, CL, DH, DL
reg16	16-bit general-purpose register: AX, BX, CX, DX, SI, DI, SP, BP
reg32	32-bit general-purpose register: EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP
reg	Any general-purpose register
sreg	16-bit segment register: CS, DS, SS, ES, FS, GS
imm	8-, 16-, or 32-bit immediate value
imm8	8-bit immediate byte value
imm16	16-bit immediate word value
imm32	32-bit immediate doubleword value
reg/mem8	8-bit operand, which can be an 8-bit general register or memory byte
reg/mem16	16-bit operand, which can be a 16-bit general register or memory word
reg/mem32	32-bit operand, which can be a 32-bit general register or memory doubleword
mem	An 8-, 16-, or 32-bit memory operand

MOV Instruction:

It is used to move data from source operand to destination operand

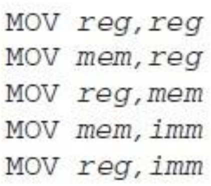
- Both operands must be the same size.
- Both operands cannot be memory operands.
- CS, EIP, and IP cannot be destination operands.

- An immediate value cannot be moved to a segment register.

Syntax:

MOV destination, source

Here is a list of the general variants of MOV, excluding segment registers:



```
MOV reg,reg
MOV mem,reg
MOV reg,mem
MOV mem,imm
MOV reg,imm
```

Example:

MOV bx, 2
MOV ax, cx

Example:

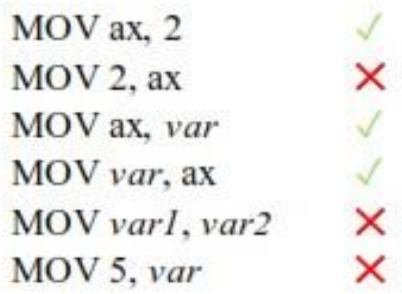
'A' has ASCII code 65D (01000001B, 41H)

The following MOV instructions stores it in register BX:

MOV bx, 65d
MOV bx, 41h
MOV bx, 01000001b
MOV bx, 'A'
All of the above are equivalent.

Examples:

The following examples demonstrate compatibility between operands used with MOV instruction:



MOV ax, 2	✓
MOV 2, ax	✗
MOV ax, *var*	✓
MOV *var*, ax	✓
MOV *var1*, *var2*	✗
MOV 5, *var*	✗

ADD Instruction

The ADD instruction adds a source operand to a destination operand of the same size. Source is unchanged by the operation, and the sum is stored in the destination operand

Syntax:

```
        ADD dest,source
```

SUB Instruction

The SUB instruction subtracts a source operand from a destination operand.

Syntax:

```
        SUB dest,source
```

Sample Program:

```
TITLE Add and Subtract (AddSub.asm)
; This program adds and subtracts 32-bit integers.
INCLUDE Irvine32.inc
.code
main PROC
mov eax,10000h          ; EAX = 10000h
add eax,40000h          ; EAX = 50000h
sub eax,20000h          ; EAX = 30000h

call DumpRegs           ; display registers
exit
main ENDP
END main
```

Lab Exercise:

1. Write an uninitialized data declaration for a16-bit signed integer val1. Initialize 8-bit signed integer val2 with -10.
2. Declare a 32-bit signed integer val3 and initialize it with the smallest possible negative decimal value. (Hint: Use SDWORD)
3. Declare an unsigned 16-bit integer variable named wArray that uses three Initializers.

4. Declare a string variable containing the name of your favorite color. Initialize it as a null terminated string. Initialize five 16-bit unsigned integers varA, varB, varC, varD & varE with the following values: 12, 2, 13, 8, 14.
5. Convert the following high-level instruction into Assembly Language:

   ebx = { (a+b) – (a-b) + c } +d a=
   10h , b=15h, c=20h, d=30h
6. Convert the given values of a,b,c,d into binary and then use in 8-bit data definition and implement in the equation.
7. Write a program in assembly language that implements following expression: Eax = imm8 + data1 – data3 + imm8 + data2

   Use these data definitions:

   Imm8 = 20

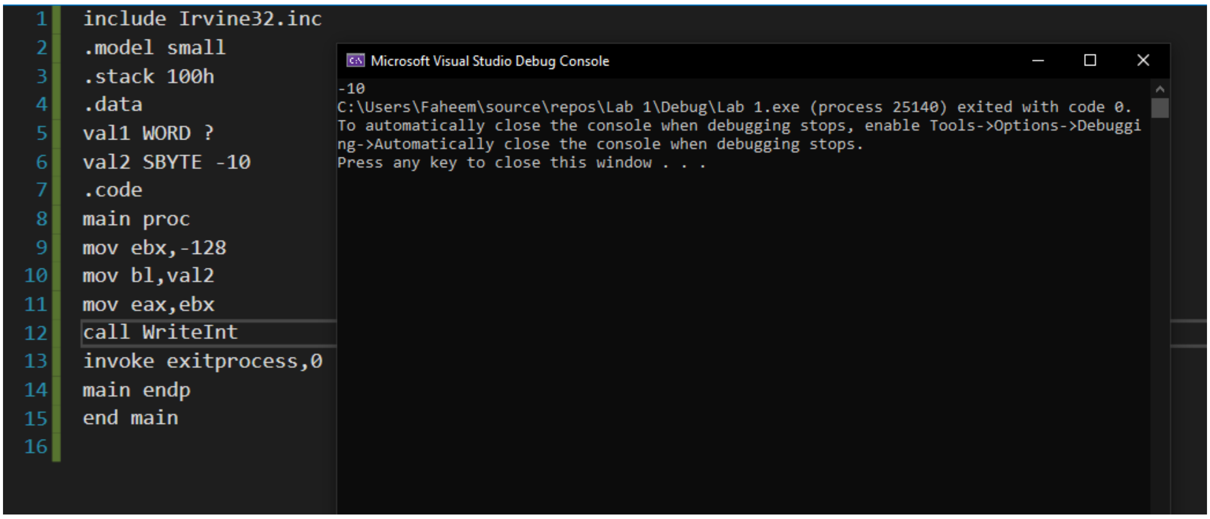
   Data1 word 8

   Data2 word 15

   Data3 word 20

COAL Lab 3 Tasks

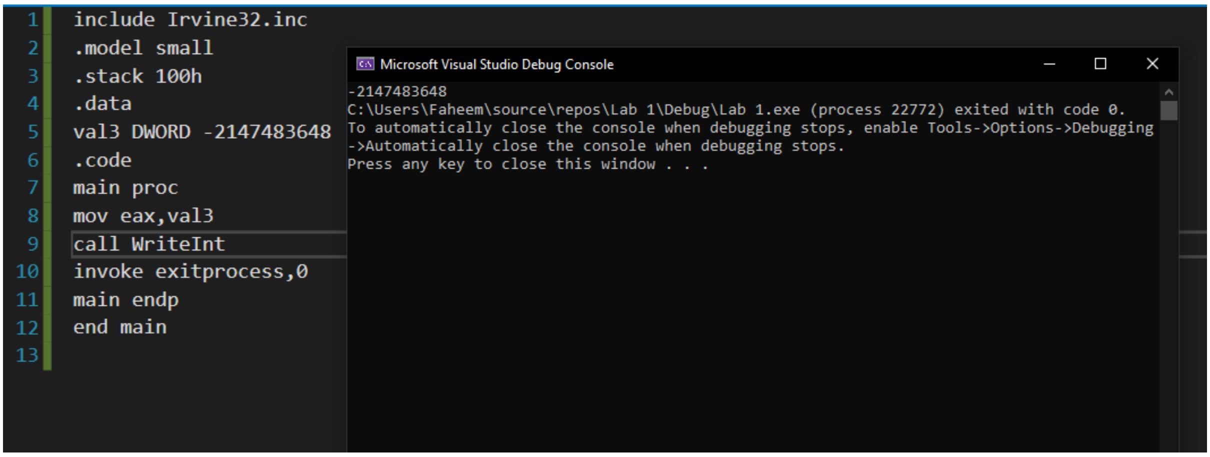
Task # 1:



```
include Irvine32.inc
.model small
.stack 100h
.data
val1 WORD ?
val2 SBYTE -10
.code
main proc
mov ebx,-128
mov bl,val2
mov eax,ebx
call WriteInt
invoke exitprocess,0
main endp
end main
```

```
-10
C:\Users\Faheem\source\repos\Lab 1\Debug\Lab 1.exe (process 25140) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debuggi
ng->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

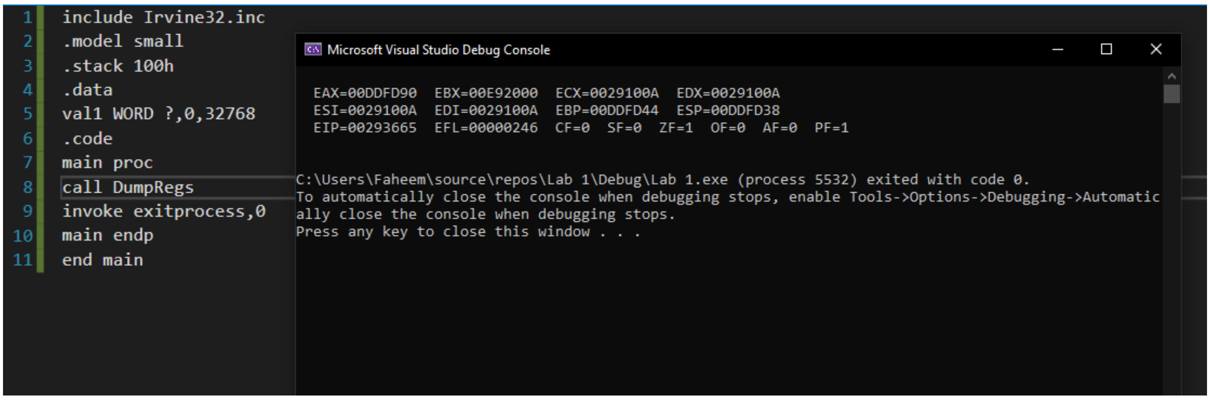
Task # 2:



```
include Irvine32.inc
.model small
.stack 100h
.data
val3 DWORD -2147483648
.code
main proc
mov eax,val3
call WriteInt
invoke exitprocess,0
main endp
end main
```

```
-2147483648
C:\Users\Faheem\source\repos\Lab 1\Debug\Lab 1.exe (process 22772) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging
->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Task # 3:

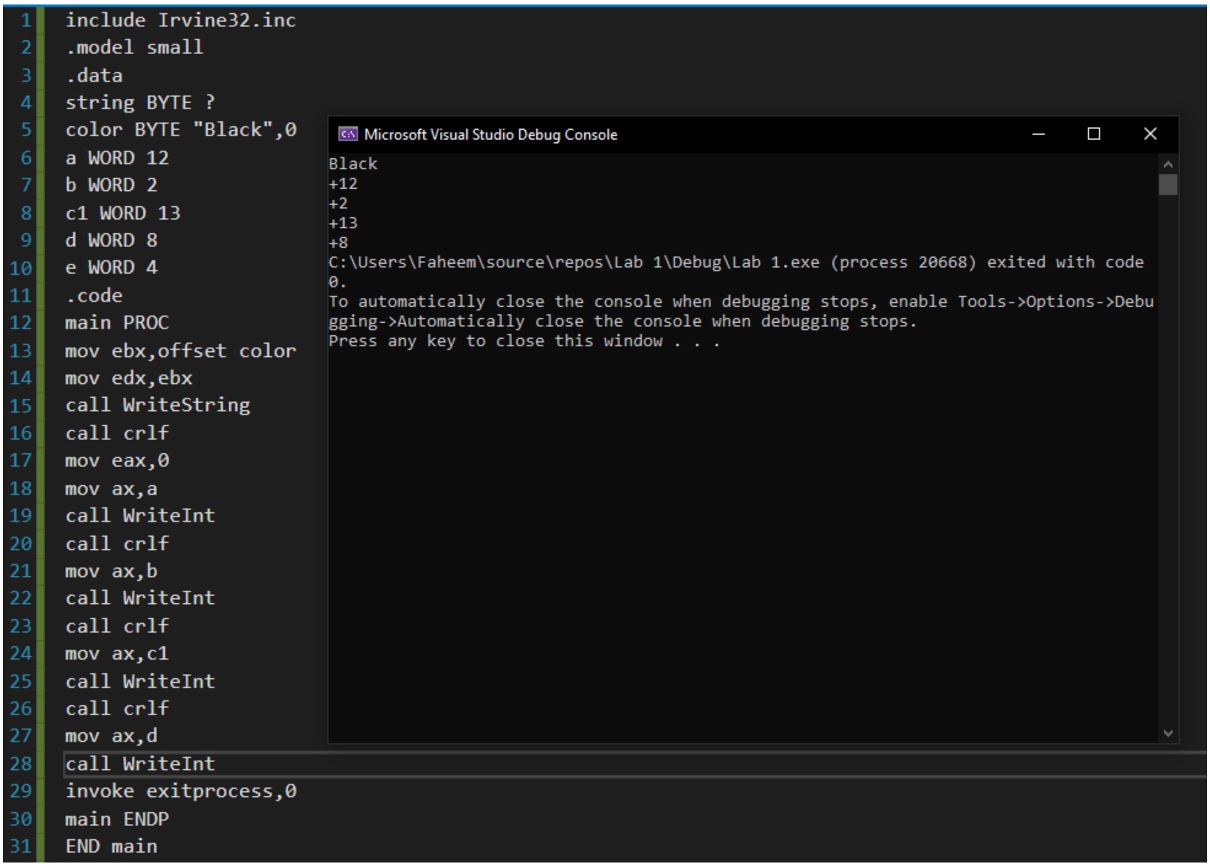


```
include Irvine32.inc
.model small
.stack 100h
.data
val1 WORD ?,0,32768
.code
main proc
call DumpRegs
invoke exitprocess,0
main endp
end main
```

```
 EAX=00DDFD90  EBX=00E92000  ECX=0029100A  EDX=0029100A
 ESI=0029100A  EDI=0029100A  EBP=00DDFD44  ESP=00DDFD38
 EIP=00293665  EFL=00000246  CF=0  SF=0  ZF=1  OF=0  AF=0  PF=1


C:\Users\Faheem\source\repos\Lab 1\Debug\Lab 1.exe (process 5532) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatic
ally close the console when debugging stops.
Press any key to close this window . . .
```

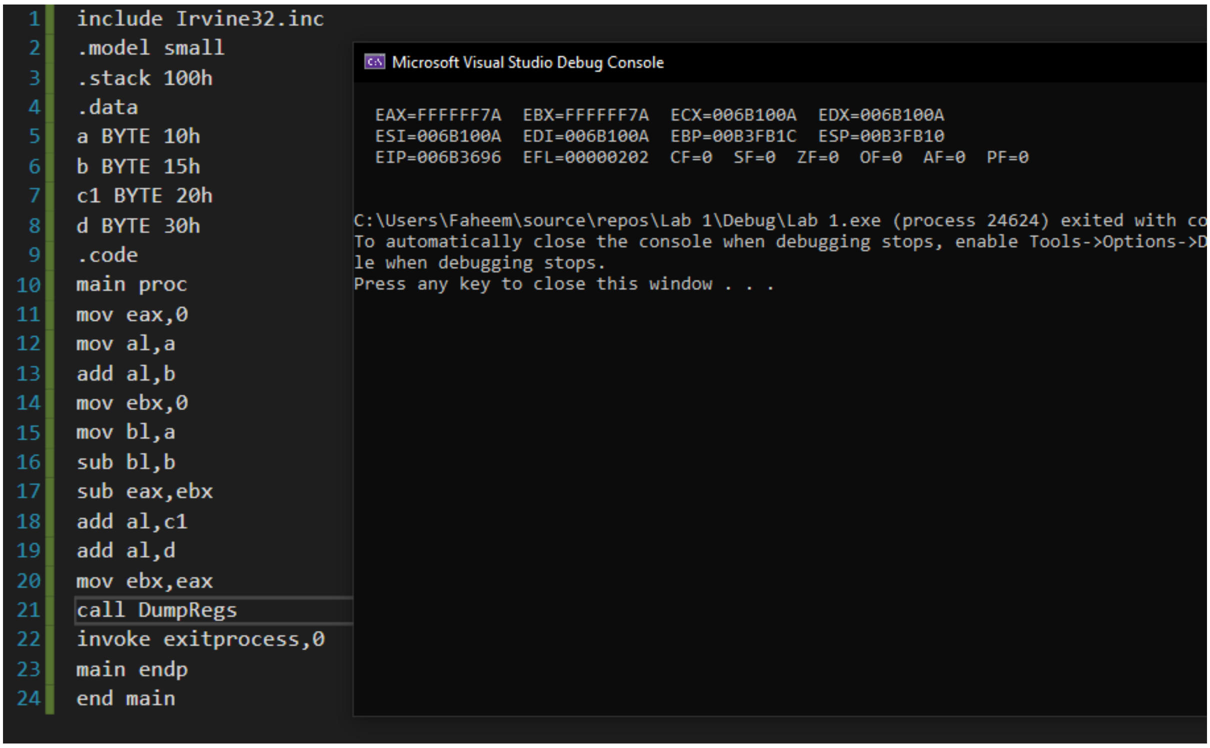
Task # 4:



```
include Irvine32.inc
.model small
.data
string BYTE ?
color BYTE "Black",0
a WORD 12
b WORD 2
c1 WORD 13
d WORD 8
e WORD 4
.code
main PROC
mov ebx,offset color
mov edx,ebx
call WriteString
call crlf
mov eax,0
mov ax,a
call WriteInt
call crlf
mov ax,b
call WriteInt
call crlf
mov ax,c1
call WriteInt
call crlf
mov ax,d
call WriteInt
invoke exitprocess,0
main ENDP
END main
```

```
Black
+12
+2
+13
+8
C:\Users\Faheem\source\repos\Lab 1\Debug\Lab 1.exe (process 20668) exited with code
0.
To automatically close the console when debugging stops, enable Tools->Options->Debu
gging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Task # 5:

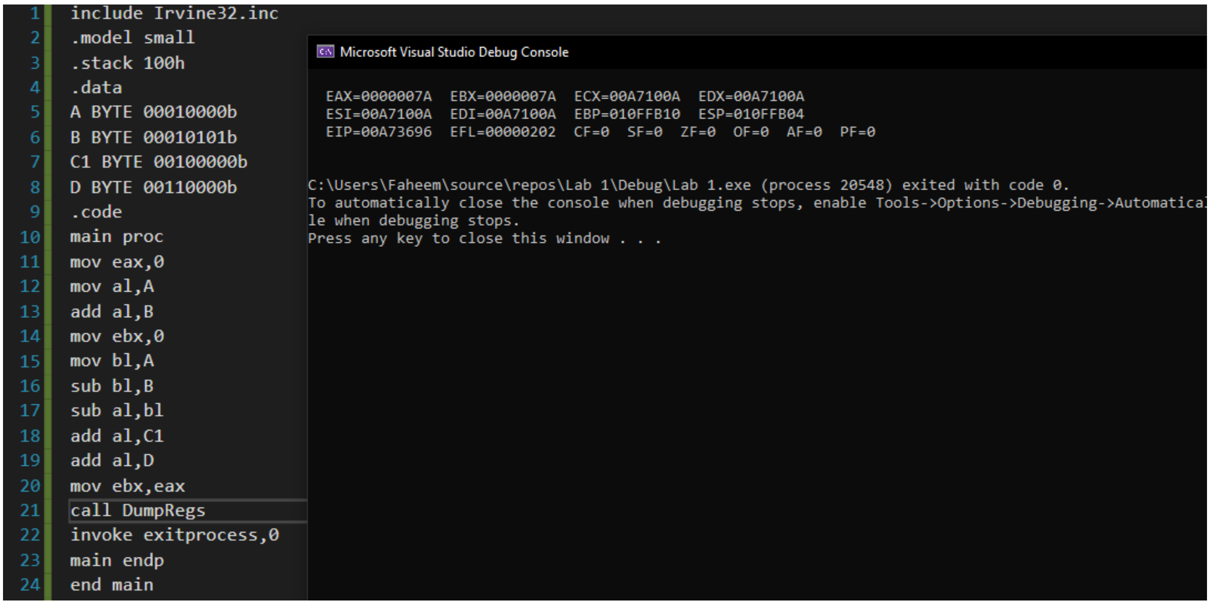


```
include Irvine32.inc
.model small
.stack 100h
.data
a BYTE 10h
b BYTE 15h
c1 BYTE 20h
d BYTE 30h
.code
main proc
mov eax,0
mov al,a
add al,b
mov ebx,0
mov bl,a
sub bl,b
sub eax,ebx
add al,c1
add al,d
mov ebx,eax
call DumpRegs
invoke exitprocess,0
main endp
end main
```

```
  EAX=FFFFFF7A  EBX=FFFFFF7A  ECX=006B100A  EDX=006B100A
  ESI=006B100A  EDI=006B100A  EBP=00B3FB1C  ESP=00B3FB10
  EIP=006B3696  EFL=00000202  CF=0  SF=0  ZF=0  OF=0  AF=0  PF=0


C:\Users\Faheem\source\repos\Lab 1\Debug\Lab 1.exe (process 24624) exited with co
To automatically close the console when debugging stops, enable Tools->Options->D
le when debugging stops.
Press any key to close this window . . .
```

Task # 6:

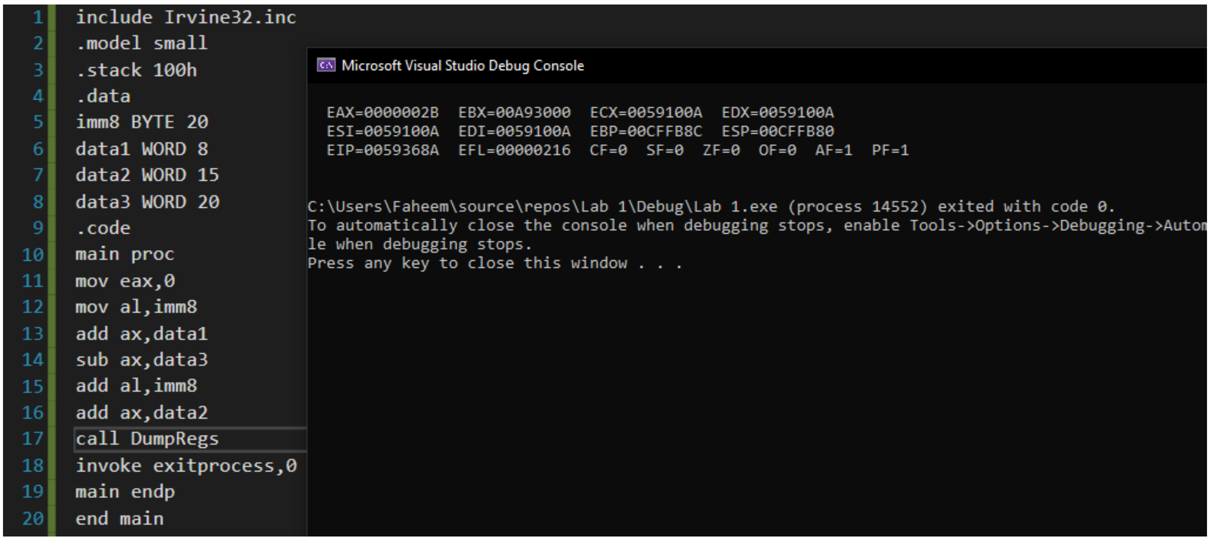


```
include Irvine32.inc
.model small
.stack 100h
.data
A BYTE 00010000b
B BYTE 00010101b
C1 BYTE 00100000b
D BYTE 00110000b
.code
main proc
mov eax,0
mov al,A
add al,B
mov ebx,0
mov bl,A
sub bl,B
sub al,bl
add al,C1
add al,D
mov ebx,eax
call DumpRegs
invoke exitprocess,0
main endp
end main
```

```
  EAX=0000007A  EBX=0000007A  ECX=00A7100A  EDX=00A7100A
  ESI=00A7100A  EDI=00A7100A  EBP=010FFB10  ESP=010FFB04
  EIP=00A73696  EFL=00000202  CF=0  SF=0  ZF=0  OF=0  AF=0  PF=0


C:\Users\Faheem\source\repos\Lab 1\Debug\Lab 1.exe (process 20548) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatica
le when debugging stops.
Press any key to close this window . . .
```

Task # 7:



```
include Irvine32.inc
.model small
.stack 100h
.data
imm8 BYTE 20
data1 WORD 8
data2 WORD 15
data3 WORD 20
.code
main proc
mov eax,0
mov al,imm8
add ax,data1
sub ax,data3
add al,imm8
add ax,data2
call DumpRegs
invoke exitprocess,0
main endp
end main
```

```
  EAX=0000002B  EBX=00A93000  ECX=0059100A  EDX=0059100A
  ESI=0059100A  EDI=0059100A  EBP=00CFFB8C  ESP=00CFFB80
  EIP=0059368A  EFL=00000216  CF=0  SF=0  ZF=0  OF=0  AF=1  PF=1


C:\Users\Faheem\source\repos\Lab 1\Debug\Lab 1.exe (process 14552) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Autom
le when debugging stops.
Press any key to close this window . . .
```